CPSC 330
Applied Machine Learning

Lecture 2: Terminology, Baselines, Decision Trees

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

In [1]:
```
%load_ext autoreload
%autoreload 2

import glob
import os
import re
import sys
from collections import Counter, defaultdict

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

sys.path.append("../code/.")
import graphviz
import IPython
from IPython.display import HTML, display
from plotting_functions import *
from sklearn.dummy import DummyClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor,
from utils import *

plt.rcParams["font.size"] = 16
pd.set_option("display.max_colwidth", 200)

```

Announcements

- Reminder of first deliverables:

  - Syllabus quiz: Jan 16, 11:59pm
  - Homework 1 (hw1): Jan 16, 11:59pm
- You can find the tentative due dates for all deliverables here (https://github.com/UBC-CS/cpsc330-2022W2#lecture-schedule-tentative).
- Please monitor Piazza (https://piazza.com/class/lcgo6c2ncl06el) (especially pinned posts and instructor posts) for announcements.
- Let's check your status: https://www.menti.com/alwcbyyjf26j (https://www.menti.com/alwcbyyjf26j)

Quick recap: True or False?

(Same menti as before: https://www.menti.com/alwcbyyjf26j (https://www.menti.com/alwcbyyjf26j))

- There are different types of machine learning problems.
- Predicting spam and predicting housing prices are both examples of supervised machine learning.
- For problems such as spelling correction, translation, face recognition, spam identification, if you are a domain expert, it's usually faster and scalable to come up with a robust set of rules manually rather than building a machine learning model.
- Google News is likely using machine learning to organize news.

Learning outcomes

From this lecture, you will be able to

Terminology [video (https://youtu.be/YNT8n4cXu4A)]

- identify whether a given problem could be solved using supervised machine learning or not;
- differentiate between supervised and unsupervised machine learning;
- explain machine learning terminology such as features, targets, predictions, training, and error;
- differentiate between classification and regression problems;

Baselines [video (https://youtu.be/6eT5cLL-2Vc)]

- use `DummyClassifier` and `DummyRegressor` as baselines for machine learning problems;
- explain the `fit` and `predict` paradigm and use `score` method of ML models;

Decision trees [video (https://youtu.be/Hcf19lj35rA)]

- broadly describe how decision tree prediction works;
- use `DecisionTreeClassifier` and `DecisionTreeRegressor` to build decision trees using `scikit-learn`;

- visualize decision trees;

More terminology [video (https://youtu.be/KEtsfXn4w2E)]

- explain the difference between parameters and hyperparameters;
- explain the concept of decision boundaries;
- explain the relation between model complexity and decision boundaries.

Big picture and datasets

In this lecture, we'll talk about our first machine learning model: Decision trees. We will also familiarize ourselves with some common terminology in supervised machine learning.

Toy datasets

Later in the course we will use larger datasets from Kaggle, for instance. But for our first couple of lectures, we will be working with the following three toy datasets:

- Quiz2 grade prediction classification dataset (data/quiz2-grade-toy-classification.csv)
- Quiz2 grade prediction regression dataset (data/quiz2-grade-toy-regression.csv)
- Canada USA cities dataset (data/canada_usa_cities.csv)

If it's not necessary for you to understand the code, it will be in one of the files under the `code` directory to avoid clutter in this notebook. For example, most of the plotting code is going to be in `code/plotting_functions.py`.

Terminology [video (https://youtu.be/YNT8n4cXu4A)]

You will see a lot of variable terminology in machine learning and statistics. Let's familiarize ourselves with some of the basic terminology used in ML.

Check out the accompanying video (https://youtu.be/YNT8n4cXu4A) on this material.

We'll be using the following grade prediction toy dataset to demonstrate the terminology. Imagine that you are taking a course with four homework assignments and two quizzes. You and your friends are quite nervous about your quiz2 grades and you want to know how will you do based on your previous performance and some other attributes. So you decide to collect some data from your friends from last year and train a supervised machine learning model for quiz2 grade prediction.

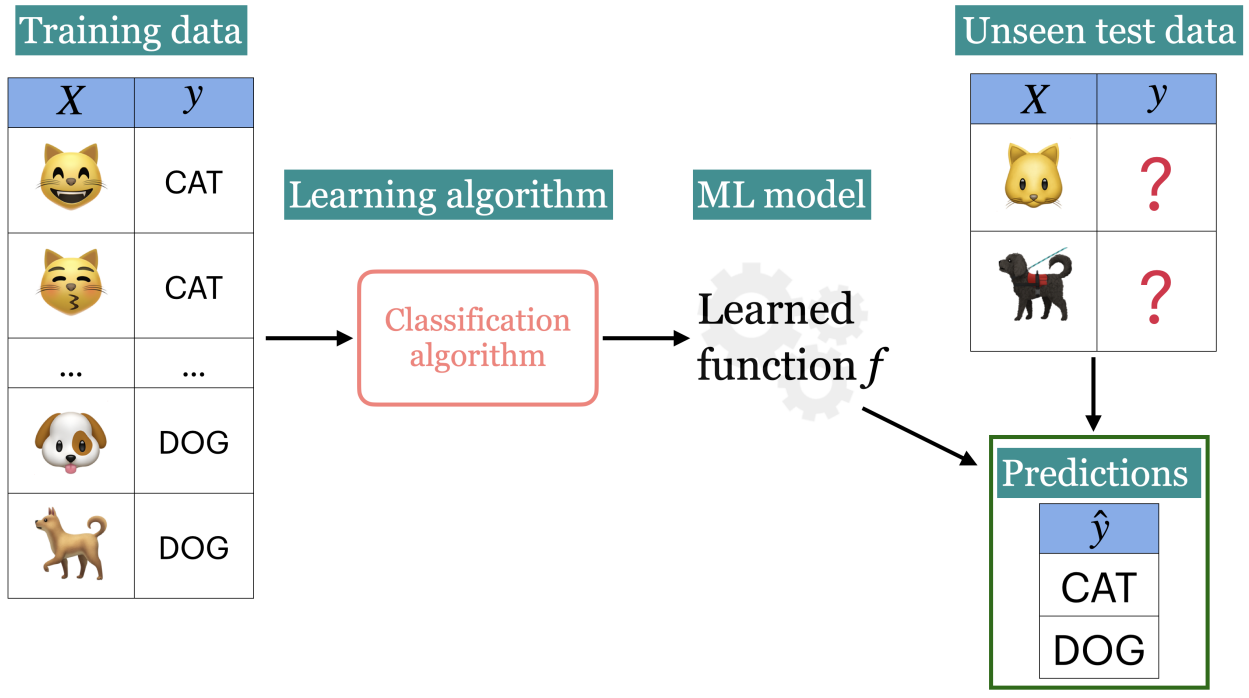
In [2]:
```
classification_df = pd.read_csv("../data/quiz2-grade-toy-classification
print(classification_df.shape)
classification_df.head()
```

```
(21, 8)
```

Out[2]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	A+
1	1	0	94	90	80	83	91	not A+
2	0	0	78	85	83	80	80	not A+
3	0	1	91	94	92	91	89	A+
4	0	1	77	83	90	92	85	A+

Recap: Supervised machine learning



Tabular data

In supervised machine learning, the input data is typically organized in a **tabular** format, where rows are **examples** and columns are **features**. One of the columns is typically the **target**.

	X Features (d)						y Target
Examples (n)	**ml_ experience**	**class_ attendance**	**lab1**	**lab2**	**lab3**	**lab4**	**quiz1**	**quiz2**
	1	1	91	93	88	92	94	A+
	1	0	78	87	88	85	80	not A+
	...	...	...	...	...	...	...	...
	0	1	69	75	65	80	65	not A+

Features : Features are relevant characteristics of the problem, usually suggested by experts. Features are typically denoted by X and the number of features is usually denoted by d.

Target : Target is the feature we want to predict (typically denoted by y).

Example : A row of feature values. When people refer to an example, it may or may not include the target corresponding to the feature values, depending upon the context. The number of examples is usually denoted by n.

Training : The process of learning the mapping between the features (X) and the target (y).

Example: Tabular data for grade prediction

The tabular data usually contains both: the features (`X`) and the target (`y`).

In [3]:
```
classification_df = pd.read_csv("../data/quiz2-grade-toy-classification
classification_df.head()
```

Out[3]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	A+
1	1	0	94	90	80	83	91	not A+
2	0	0	78	85	83	80	80	not A+
3	0	1	91	94	92	91	89	A+
4	0	1	77	83	90	92	85	A+

So the first step in training a supervised machine learning model is separating `X` and `y` .

In [4]:
```
X = classification_df.drop(columns=["quiz2"])
y = classification_df["quiz2"]
X.head()
```

Out[4]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1
0	1	1	92	93	84	91	92
1	1	0	94	90	80	83	91
2	0	0	78	85	83	80	80
3	0	1	91	94	92	91	89
4	0	1	77	83	90	92	85

In [5]:
```
y.head()
```

Out[5]:
```
0        A+
1    not A+
2    not A+
3        A+
4        A+
Name: quiz2, dtype: object
```

In [6]:
```
X.shape
```

Out[6]:
```
(21, 7)
```

Example: Tabular data for the housing price prediction

Here is an example of tabular data for housing price prediction. You can download the data from here (https://www.kaggle.com/harlfoxem/housesalesprediction).

In [7]:
```
housing_df = pd.read_csv("../data/kc_house_data.csv")
housing_df.drop(["id", "date"], axis=1, inplace=True)
HTML(housing_df.head().to_html(index=False))
```

Out[7]:

price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft
221900.0	3	1.00	1180	5650	1.0	0	0	3	7	
538000.0	3	2.25	2570	7242	2.0	0	0	3	7	
180000.0	2	1.00	770	10000	1.0	0	0	3	6	
604000.0	4	3.00	1960	5000	1.0	0	0	5	7	
510000.0	3	2.00	1680	8080	1.0	0	0	3	8	

In [8]:
```
X = housing_df.drop(columns=["price"])
y = housing_df["price"]
X.head()
```

Out[8]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above
0	3	1.00	1180	5650	1.0	0	0	3	7	1180
1	3	2.25	2570	7242	2.0	0	0	3	7	2170
2	2	1.00	770	10000	1.0	0	0	3	6	770
3	4	3.00	1960	5000	1.0	0	0	5	7	1050
4	3	2.00	1680	8080	1.0	0	0	3	8	1680

In [9]:
```
y.head()
```

Out[9]:
```
0    221900.0
1    538000.0
2    180000.0
3    604000.0
4    510000.0
Name: price, dtype: float64
```

To a machine, column names (features) have no meaning. Only feature values and how they vary across examples mean something.

Alternative terminology for examples, features, targets, and training

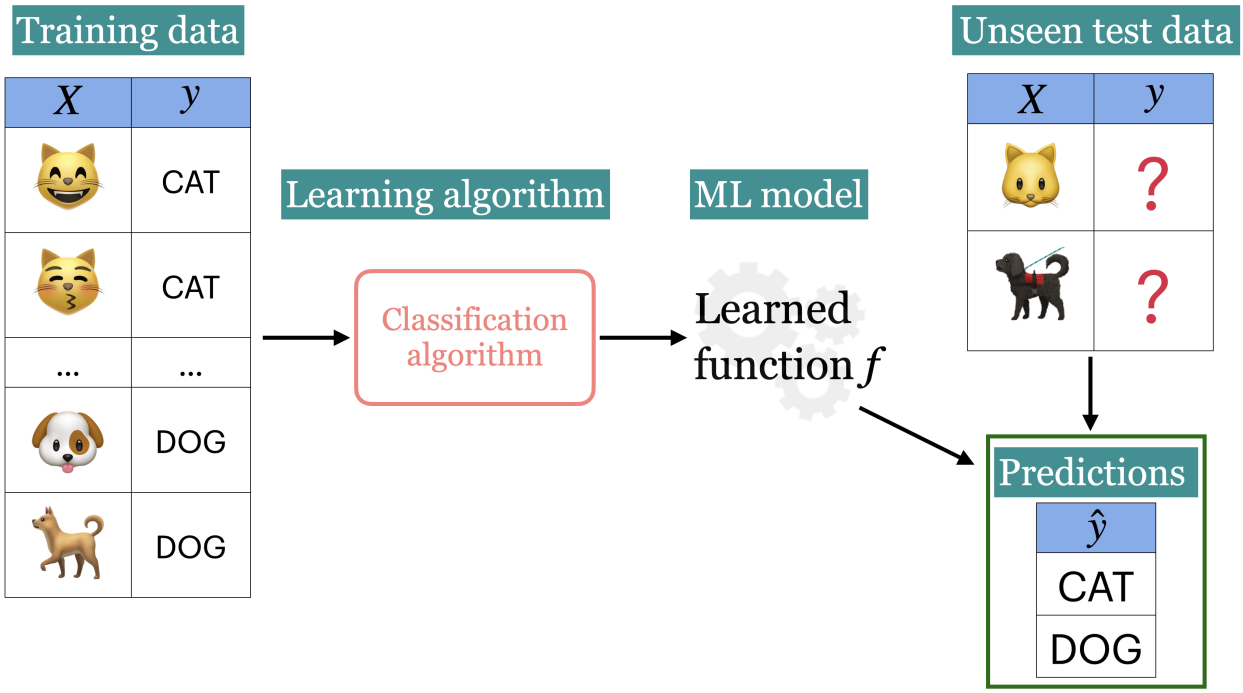
- **examples** = rows = samples = records = instances
- **features** = inputs = predictors = explanatory variables = regressors = independent variables = covariates
- **targets** = outputs = outcomes = response variable = dependent variable = labels (if categorical).
- **training** = learning = fitting

Check out [the MDS terminology document (https://ubc-mds.github.io/resources_pages/terminology/)](https://ubc-mds.github.io/resources_pages/terminology/).

Let's pause our grades investigation for a second and think about the content of video 2.1 (https://youtu.be/YNT8n4cXu4A). In this video, you learned about the differences between unsupervised and supervised learning, and between classification and regression. The following slides are a quick recap.

Supervised learning vs. Unsupervised learning

In **supervised learning**, training data comprises a set of features (X) and their corresponding targets (y). We wish to find a **model function** f that relates X to y. Then use that model function **to predict the targets** of new examples.



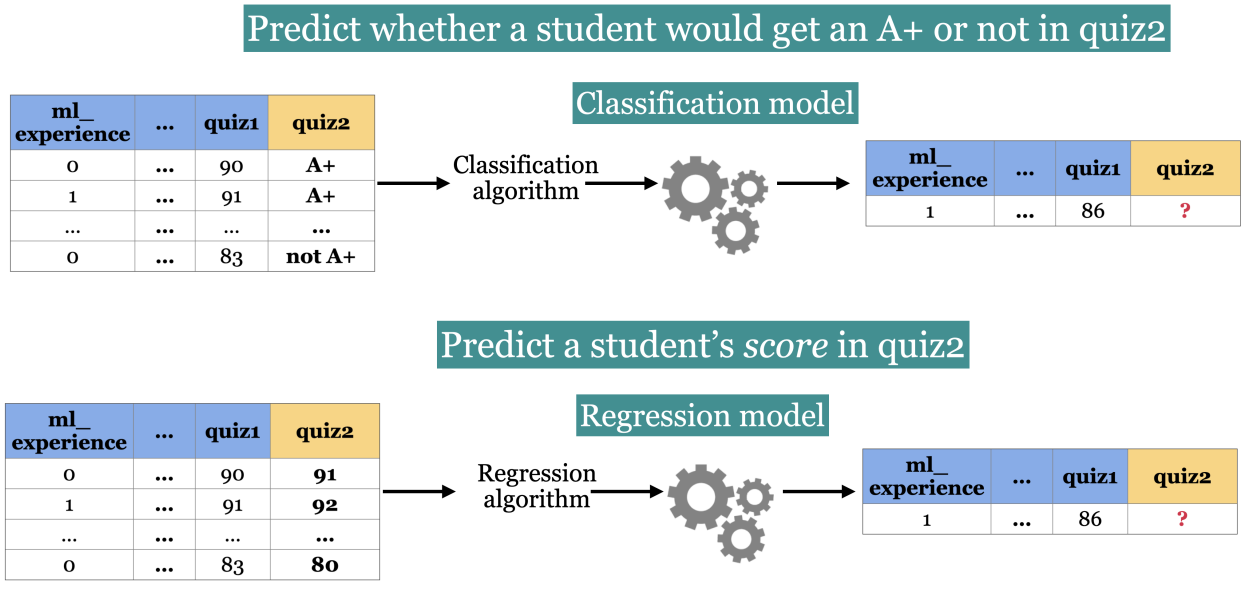
In **unsupervised learning** training data consists of observations (X) **without any corresponding targets**. Unsupervised learning could be used to **group similar things together** in X or to provide **concise summary** of the data. We'll learn more about this topic in later videos.

Supervised machine learning is about function approximation, i.e., finding the mapping function between `X` and `y` whereas unsupervised machine learning is about concisely describing the data.

Classification vs. Regression

In supervised machine learning, there are two main kinds of learning problems based on what they are trying to predict.

- **Classification problem**: predicting among two or more discrete classes
  - Example1: Predict whether a patient has a liver disease or not
  - Example2: Predict whether a student would get an A+ or not in quiz2.
- **Regression problem**: predicting a continuous value
  - Example1: Predict housing prices
  - Example2: Predict a student's score in quiz2.



In [10]:
```
# quiz2 classification toy data
classification_df = pd.read_csv("../data/quiz2-grade-toy-classification
classification_df.head(4)
```

Out[10]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	A+
1	1	0	94	90	80	83	91	not A+
2	0	0	78	85	83	80	80	not A+
3	0	1	91	94	92	91	89	A+

In [11]:
```
# quiz2 regression toy data
regression_df = pd.read_csv("../data/quiz2-grade-toy-regression.csv")
regression_df.head(4)
```

Out[11]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	90
1	1	0	94	90	80	83	91	84
2	0	0	78	85	83	80	80	82
3	0	1	91	94	92	91	89	92

? ? Questions for you

```
Which of these are examples of supervised learning?

1. Finding groups of similar properties in a real estate data set.
2. Predicting real estate prices based on house features like numbe
r of rooms, learning from past sales as examples.
3. Grouping articles on different topics from different news source
s (something like the Google News app).
4. Detecting credit card fraud based on examples of fraudulent and
non-fraudulent transactions.
```

Let's answer here: https://www.menti.com/alf1tszig9pa (https://www.menti.com/alf1tszig9pa)

```
Which of these are examples of classification and which ones are of
regression?

1. Predicting the price of a house based on features such as number
of bedrooms and the year built.
2. Predicting if a house will sell or not based on features like th
e price of the house, number of rooms, etc.
3. Predicting percentage grade in CPSC 330 based on past grades.
4. Predicting whether you should bicycle tomorrow or not based on t
he weather forecast.
```

Let's answer here: https://www.menti.com/alf1tszig9pa (https://www.menti.com/alf1tszig9pa)

Baselines [video (https://youtu.be/6eT5cLL-2Vc)]

Check out the accompanying video (https://youtu.be/6eT5cLL-2Vc) on this material.

Supervised learning (Reminder)

- Training data → Machine learning algorithm → ML model
- Unseen test data + ML model → predictions

Let's build a very simple supervised machine learning model for quiz2 grade prediction problem.

In [12]:
```
classification_df = pd.read_csv("../data/quiz2-grade-toy-classification
classification_df.head()
```

Out[12]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	A+
1	1	0	94	90	80	83	91	not A+
2	0	0	78	85	83	80	80	not A+
3	0	1	91	94	92	91	89	A+
4	0	1	77	83	90	92	85	A+

In [13]:
```
classification_df['quiz2'].value_counts()
```

Out[13]:
```
not A+    11
A+        10
Name: quiz2, dtype: int64
```

Seems like "not A+" occurs more frequently than "A+". What if we predict "not A+" all the time?

Baselines

Baseline : A simple machine learning algorithm based on simple rules of thumb.

- For example, most frequent baseline always predicts the most frequent label in the training set.
- Baselines provide a way to sanity check your machine learning model.

`DummyClassifier`

- `sklearn` 's baseline model for classification
- Let's train `DummyClassifier` on the grade prediction dataset.

Steps to train a classifier using `sklearn`

1. Read the data
2. Create X and y
3. Create a classifier object
4. `fit` the classifier
5. `predict` on new examples
6. `score` the model

Reading the data

In [14]:
```
classification_df.head()
```

Out[14]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	A+
1	1	0	94	90	80	83	91	not A+
2	0	0	78	85	83	80	80	not A+
3	0	1	91	94	92	91	89	A+
4	0	1	77	83	90	92	85	A+

Create X and y

- $X \rightarrow$ Feature vectors
- $y \rightarrow$ Target

In [15]:
```
X = classification_df.drop(columns=["quiz2"])
y = classification_df["quiz2"]
```

Create a classifier object

- `import` the appropriate classifier
- Create an object of the classifier

In [16]:
```
from sklearn.dummy import DummyClassifier # import the classifier

dummy_clf = DummyClassifier(strategy="most_frequent") # Create a classi
```

`fit` the classifier

- The "learning" is carried out when we call `fit` on the classifier object.

In [17]:
```
dummy_clf.fit(X, y); # fit the classifier
```

`predict` the target of given examples

- We can predict the target of examples by calling `predict` on the classifier object.

In [18]:
```
dummy_clf.predict(X) # predict using the trained classifier
```

Out[18]:
```
array(['not A+', 'not A+', 'not A+', 'not A+', 'not A+', 'not A+',
       'not A+', 'not A+', 'not A+', 'not A+', 'not A+', 'not A+',
       'not A+', 'not A+', 'not A+', 'not A+', 'not A+', 'not A+',
       'not A+', 'not A+', 'not A+'], dtype='<U6')
```

`score` your model

- How do you know how well your model is doing?
- For classification problems, by default, `score` gives the **accuracy** of the model, i.e., proportion of correctly predicted targets.

$$accuracy = \frac{\text{correct predictions}}{\text{total examples}}$$

In [19]:
```
print("The accuracy of the model on the training data: %0.3f" % (dummy_
```

```
The accuracy of the model on the training data: 0.524
```

- Sometimes you will also see people reporting **error**, which is usually $1 - accuracy$
- `score`
  - calls `predict` on `X`
  - compares predictions with `y` (true targets)
  - returns the accuracy in case of classification.

In [20]:
```
print(
    "The error of the model on the training data: %0.3f" % (1 - dummy_c
)
```

```
The error of the model on the training data: 0.476
```

`fit`, `predict`, and `score` summary

Here is the general pattern when we build ML models using `sklearn`.

In [21]:
```
# Create `X` and `y` from the given data
X = classification_df.drop(columns=["quiz2"])
y = classification_df["quiz2"]

clf = DummyClassifier(strategy="most_frequent") # Create a class object
clf.fit(X, y) # Train/fit the model
print(clf.score(X, y)) # Assess the model

new_examples = [[0, 1, 92, 90, 95, 93, 92], [1, 1, 92, 93, 94, 92]]
clf.predict(new_examples) # Predict on some new data using the trained
```

```
0.5238095238095238
```

Out[21]:
```
array(['not A+', 'not A+'], dtype='<U6')
```

`DummyRegressor` (https://scikit-learn.org/0.15/modules/generated/sklearn.dummy.DummyRegressor.html

You can also do the same thing for regression problems using `DummyRegressor`.

If the DummyClassifier picked the most "popular" target, what should the DummyRegressor Pick?

- Let's build a regression baseline model using `sklearn`.

In [22]:
```
from sklearn.dummy import DummyRegressor

regression_df = pd.read_csv("../data/quiz2-grade-toy-regression.csv") #
X = regression_df.drop(columns=["quiz2"]) # Create `X` and `y` from the
y = regression_df["quiz2"]
reg = DummyRegressor() # Create a class object; uses mean as default st
reg.fit(X, y) # Train/fit the model
print(reg.score(X, y)) # Assess the model
new_examples = [[0, 1, 92, 90, 95, 93, 92], [1, 1, 92, 93, 94, 92]]
reg.predict(new_examples) # Predict on some new data using the trained
```

```
0.0
```

Out[22]:
```
array([86.28571429, 86.28571429])
```

- The `fit` and `predict` paradigms similar to classification. The `score` method in the context of regression returns somethings called R^2 score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score). (More on this in later videos.)
  - The maximum R^2 is 1 for perfect predictions.
  - For `DummyRegressor` it returns the mean of the `y` values.

In [23]:
```
reg.score(X, y)
```

Out[23]:
```
0.0
```

? ? Questions for you

```
1. Order the steps below to build ML models using `sklearn`.
    - `score` to evaluate the performance of a given model
    - `predict` on new examples
    - Creating a model instance
    - Creating `X` and `y`
    - `fit`
2. Why do we train a Dummy Classifier?
```

Let's answer here: https://www.menti.com/alvzb1o23znq (https://www.menti.com/alvzb1o23znq)

Decision trees [video (https://youtu.be/Hcf19lj35rA)]

Check out the accompanying video (https://youtu.be/Hcf19lj35rA) on this material.

Writing a traditional program to predict quiz2 grade

- Can we do better than the baseline?
- Forget about ML for a second. If you are asked to write a program to predict whether a student gets an A+ or not in quiz2, how would you go for it?
- For simplicity, let's binarize the feature values.

1 = "yes"; 0 = "no" 1 = "A+"; 0 = "not A+"

ml_ experience	class_ attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	1	0	1	1	A+
0	0	1	1	0	0	1	not A+
1	0	0	1	0	1	0	not A+
0	1	0	0	1	1	0	A+
1	0	0	1	1	1	1	A+
0	1	0	1	0	0	0	not A+

- Is there a pattern that distinguishes yes's from no's and what does the pattern say about today?
- How about a rule-based algorithm with a number of *if else* statements?

```
if class_attendance == 1 and quiz1 == 1:
    quiz2 == "A+"
elif class_attendance == 1 and lab3 == 1 and lab4 == 1:
    quiz2 == "A+"
...
```

- How many possible rule combinations there could be with the given 7 binary features?
  - Gets unwieldy pretty quickly

Decision tree algorithm

- A machine learning algorithm to derive such rules from data in a principled way.
- Have you ever played 20-questions game (https://en.wikipedia.org/wiki/Twenty_questions)? Decision trees are based on the same idea!
- Let's `fit` a decision tree using `scikit-learn` and `predict` with it.
- Recall that `scikit-learn` uses the term `fit` for training or learning and uses `predict` for prediction.

Building decision trees with `sklearn`

Let's binarize our toy dataset for simplicity.

In [24]:
```
classification_df = pd.read_csv("../data/quiz2-grade-toy-classification
X = classification_df.drop(columns=["quiz2"])
y = classification_df["quiz2"]

X_binary = X.copy()
columns = ["lab1", "lab2", "lab3", "lab4", "quiz1"]
for col in columns:
    X_binary[col] = X_binary[col].apply(lambda x: 1 if x >= 90 else 0)
X_binary.head()
```

Out[24]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1
0	1	1	1	1	0	1	1
1	1	0	1	1	0	0	1
2	0	0	0	0	0	0	0
3	0	1	1	1	1	1	0
4	0	1	0	0	1	1	0

In [25]:
```
y.head()
```

Out[25]:
```
0        A+
1    not A+
2    not A+
3        A+
4        A+
Name: quiz2, dtype: object
```

`DummyClassifier` on quiz2 grade prediction toy dataset

In [26]:
```
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(X_binary, y)
dummy_clf.score(X_binary, y)
```

Out[26]: 0.5238095238095238

`DecisionTreeClassifier` on quiz2 grade prediction toy dataset

In [27]:
```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier() # Create a decision tree
model.fit(X_binary, y) # Fit a decision tree
model.score(X_binary, y) # Assess the model
```

Out[27]: 0.9047619047619048

The decision tree classifier is giving much higher accuracy than the dummy classifier. That's good news!

In [28]:
```
display_tree(X_binary.columns, model, counts=True) # model visualizatic
```

Out[28]: <graphviz.sources.Source at 0x104c1fdc0>

In [30]:
```
classification_df[
    (classification_df["lab4"]>=90)
    & (classification_df["class_attendance"]> .5)
    & (classification_df["lab2"]>=90)
    & (classification_df["quiz1"]<90)
    # & (classification_df["ml_experience"]> .5)
]
```

Out[30]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
3	0	1	91	94	92	91	89	A+
9	1	1	95	95	94	94	85	not A+
14	0	1	95	90	93	95	70	not A+
20	1	1	96	92	92	96	87	A+

Some terminology related to trees

Here is a commonly used terminology in a typical representation of decision trees.

A root node : represents the first condition to check or question to ask

A branch : connects a node (condition) to the next node (condition) in the tree. Each branch typically represents either true or false.

An internal node : represents conditions within the tree

A leaf node : represents the predicted class/value when the path from root to the leaf node is followed.

How does `predict` work?

In [28]:
```
new_example = np.array([[0, 1, 0, 0, 1, 1, 1]])
new_example = pd.DataFrame(data=new_example, columns=X.columns)
new_example
```

Out[28]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1
0	0	1	0	0	1	1	1

In [52]:
```
display_tree(X_binary.columns, model)
```

Out[52]:
```
<graphviz.sources.Source at 0x1870424a0>
```

What's the prediction for the new example?

In [30]:
```
model.predict(new_example)
```

Out[30]:
```
array(['A+'], dtype=object)
```

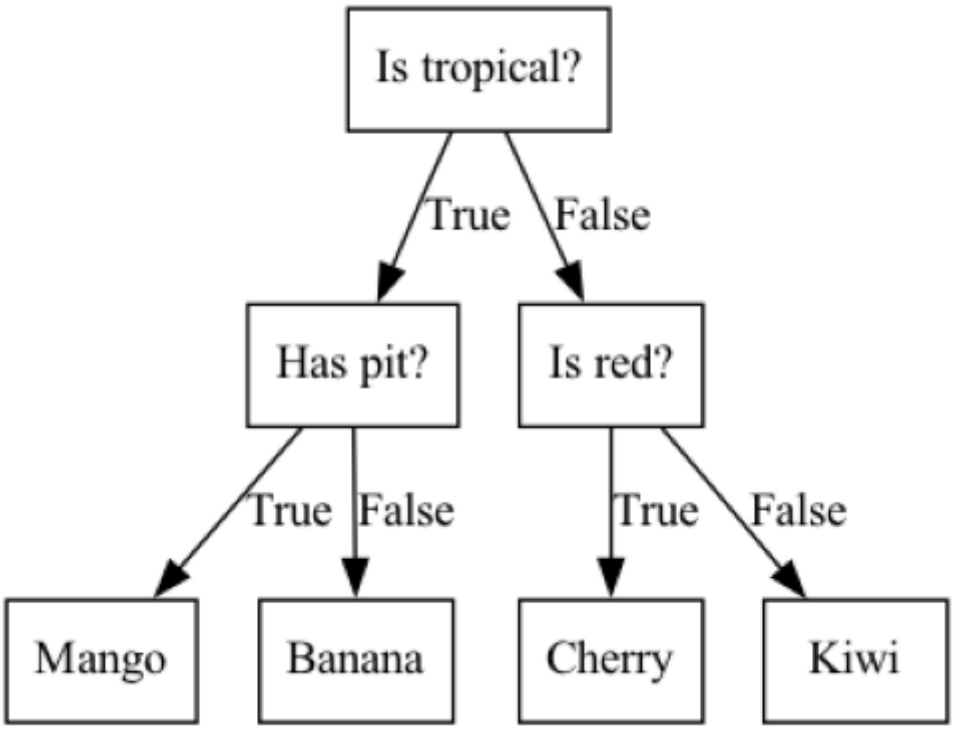
In summary, given a learned tree and a test example, during prediction time,

- Start at the top of the tree. Ask binary questions at each node and follow the appropriate path in the tree. Once you are at a leaf node, you have the prediction.
- Note that the model only considers the features which are in the learned tree and ignores all other features.

How does `fit` work?

- Decision tree is inspired by 20-questions game (https://en.wikipedia.org/wiki/Twenty_questions).
- Each node either represents a question or an answer. The terminal nodes (called leaf nodes) represent answers.

In [31]:
```
1 plot_fruit_tree()
```



How does `fit` work?

- Which features are most useful for classification?
- Minimize **impurity** at each question
- Common criteria to minimize impurity: gini index (https://scikit-learn.org/stable/modules/tree.html#classification-criteria), information gain, cross entropy

In [36]:
```
1 from sklearn.tree import DecisionTreeClassifier
2
3 model = DecisionTreeClassifier() # Create a decision tree
4 model.fit(X_binary, y) # Fit a decision tree
5 display_tree(X_binary.columns, model)
```

Out[36]:
```
<graphviz.sources.Source at 0x1879b90f0>
```

We won't go through **how** it does this - that's CPSC 340. But it's worth noting that it support two types of inputs: 1. Categorical (e.g., Yes/No or more options, as shown in the tree above) 2. Numeric (a number)In the numeric case, the decision tree algorithm also picks the *threshold*.

Decision trees with continuous features

In [33]:
```
X.head()
```

Out[33]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1
0	1	1	92	93	84	91	92
1	1	0	94	90	80	83	91
2	0	0	78	85	83	80	80
3	0	1	91	94	92	91	89
4	0	1	77	83	90	92	85

In [51]:
```
model = DecisionTreeClassifier()
model.fit(X, y)
display_tree(X.columns, model)
```

Out[51]:
```
<graphviz.sources.Source at 0x10a4620e0>
```

Decision tree for regression problems

- We can also use decision tree algorithm for regression.
- Instead of gini, we use some other criteria (https://scikit-learn.org/stable/modules/tree.html#mathematical-formulation) for splitting. A common one is mean squared error (MSE). (More on this in later videos.)
- `scikit-learn` supports regression using decision trees with `DecisionTreeRegressor`
  - `fit` and `predict` paradigms similar to classification
  - `score` returns somethings called R^2 score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score
    - The maximum R^2 is 1 for perfect predictions.
    - It can be negative which is very bad (worse than `DummyRegressor`).

In [37]:
```
regression_df = pd.read_csv("../data/quiz2-grade-toy-regression.csv")
regression_df.head()
```

Out[37]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	92	93	84	91	92	90
1	1	0	94	90	80	83	91	84
2	0	0	78	85	83	80	80	82
3	0	1	91	94	92	91	89	92
4	0	1	77	83	90	92	85	90

In [38]:
```
X = regression_df.drop(["quiz2"], axis=1)
y = regression_df["quiz2"]

depth = 2
reg_model = DecisionTreeRegressor(max_depth=depth)
reg_model.fit(X, y);
regression_df["predicted_quiz2"] = reg_model.predict(X)
print("R^2 score on the training data: %0.3f\n\n" % (reg_model.score(X,
regression_df.head()
```

```
R^2 score on the training data: 0.989
```

Out[38]:

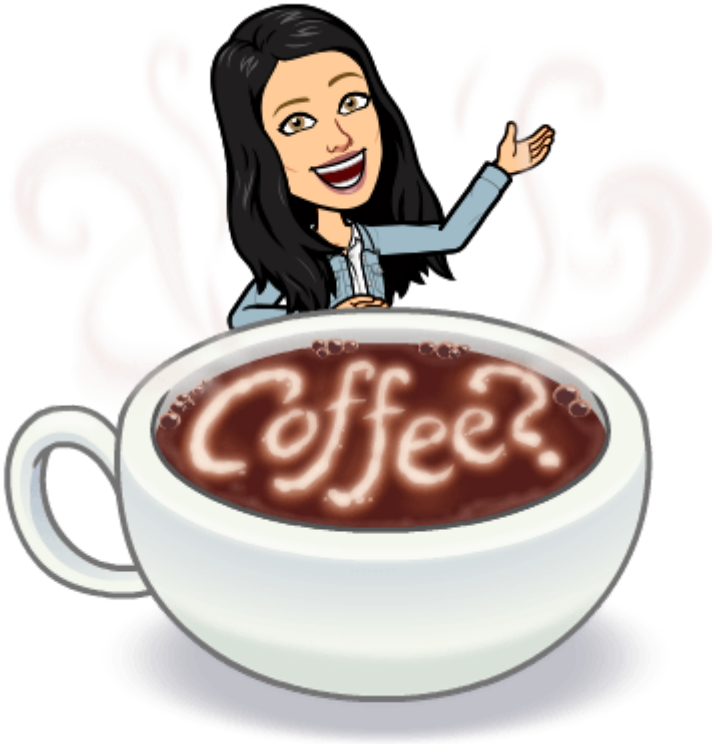
	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2	predicted_quiz2
0	1	1	92	93	84	91	92	90	90.333333
1	1	0	94	90	80	83	91	84	83.000000
2	0	0	78	85	83	80	80	82	83.000000
3	0	1	91	94	92	91	89	92	92.000000
4	0	1	77	83	90	92	85	90	90.333333

❓ ❓ True of False?

1. For the decision tree algorithm to work, the feature values must be numeric.
2. For the decision tree algorithm to work, the target values must be numeric.
3. When working with numeric features, the tree will pick the mean as threshold value.

Answer here: https://www.menti.com/albuty3ge2d4 (https://www.menti.com/albuty3ge2d4)

Break (5 min)



More terminology [video (https://youtu.be/KEtsfXn4w2E)]

- Parameters and hyperparameters
- Decision boundary

Check out the accompanying video (https://youtu.be/KEtsfXn4w2E) on this material.

Parameters

- The decision tree algorithm primarily learns two things:
  - the best feature to split on
  - the threshold for the feature to split on at each node

- These are called **parameters** of the decision tree model.
- When predicting on new examples, we need parameters of the model.

In [39]:
```
classification_df = pd.read_csv("../data/quiz2-grade-toy-classification
X = classification_df.drop(columns=["quiz2"])
y = classification_df["quiz2"]
model = DecisionTreeClassifier()
model.fit(X, y);
```

In [40]:
```
display_tree(X.columns, model, counts=True)
```

Out[40]: <graphviz.sources.Source at 0x1878a5120>

- With the default setting, the nodes are expanded until all leaves are "pure" (or further splits do not improve the classification).

- The decision tree is creating very specific rules, based on just one example from the data.
- Is it possible to control the learning in any way?
  - Yes! One way to do it is by controlling the **depth** of the tree, which is the length of the longest path from the tree root to a leaf.

Decision tree with `max_depth=1`

Decision stump : A decision tree with only one split (depth=1) is called a **decision stump**.

In [42]:
```
model = DecisionTreeClassifier(max_depth=1)
model.fit(X, y)
display_tree(X.columns, model, counts=True)
```

Out[42]: <graphviz.sources.Source at 0x1877cb6d0>

`max_depth` is a **hyperparameter** of `DecisionTreeClassifier` .

Decision tree with `max_depth=3`

In [41]:
```
model = DecisionTreeClassifier(
    max_depth=3
)  # Let's try another value for the hyperparameter
model.fit(X, y)
display_tree(X.columns, model, counts=True)
```

Out[41]: <graphviz.sources.Source at 0x18766f970>

Parameters and hyperparameters: Summary

Parameters : When you call `fit` , a bunch of values get set, like the features to split on and split thresholds. These are called **parameters**. These are learned by the algorithm from the data during training. We need them during prediction time.

Hyperparameters : Even before calling `fit` on a specific data set, we can set some "knobs" that control the learning. These are called **hyperparameters**. These are specified based on: expert knowledge, heuristics, or systematic/automated optimization (more on this in the coming lectures).

In `sklearn` hyperparameters are set in the constructor.

Above we looked at the `max_depth` hyperparameter. Some other commonly used hyperparameters of decision tree are:

- `min_samples_split`
- `min_samples_leaf`
- `max_leaf_nodes`

See [the documentation (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html) for other hyperparameters of a tree.

Decision boundary

What do we do with learned models? So far we have been using them to predict the class of a new instance. Another way to think about them is to ask: what sort of test examples will the model classify as positive, and what sort will it classify as negative?

Example 1: quiz 2 grade prediction

For visualization purposes, let's consider a subset of the data with only two features.

In [ ]:
```
X_subset = X[["lab4", "quiz1"]]
X_subset.head()
```

Decision boundary for `max_depth=1`

In [ ]:
```
depth = 1  # decision stump
model = DecisionTreeClassifier(max_depth=depth)
model.fit(X_subset.to_numpy(), y)
plot_tree_decision_boundary_and_tree(
    model, X_subset, y, x_label="lab4", y_label="quiz1"
)
```

We assume geometric view of the data. Here, the red region corresponds to "not A+" class and blue region corresponds to "A+" class. And there is a line separating the red region and the blue region which is called the **decision boundary** of the model. Different models have different kinds of decision boundaries. In decision tree models, when we are working with only two features, the decision boundary is made up of horizontal and vertical lines. In the example above, the decision boundary is created by asking one question `lab4 <= 84.5`.

Decision boundary for `max_depth=2`

In [ ]:
```
model = DecisionTreeClassifier(max_depth=2)
model.fit(X_subset.to_numpy(), y)
plot_tree_decision_boundary_and_tree(
    model, X_subset, y, x_label="lab4", y_label="quiz1"
)
```

The decision boundary, i.e., the model gets a bit more complicated.

Decision boundary for `max_depth=5`

In [ ]:
```
model = DecisionTreeClassifier(max_depth=5)
model.fit(X_subset.to_numpy(), y)
plot_tree_decision_boundary_and_tree(
    model, X_subset, y, x_label="lab4", y_label="quiz1"
)
```

The decision boundary, i.e., the model gets even more complicated with `max_depth=5`.

Example 2: Predicting country using the longitude and latitude

Imagine that you are given longitude and latitude of some border cities of USA and Canada along with which country they belong to. Using this training data, you are supposed to come up with a classification model to predict whether a given longitude and latitude combination is in the USA or Canada.

In [1]:
```
# US Canada cities data
df = pd.read_csv("../data/canada_usa_cities.csv")
df
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 2
      1 # US Canada cities data
----> 2 df = pd.read_csv("../data/canada_usa_cities.csv")
      3 df

NameError: name 'pd' is not defined
```

In [ ]:
```
X = df[["longitude", "latitude"]]
```

In [ ]:
```
y = df["country"]
```

In [ ]:
```
mglearn.discrete_scatter(X.iloc[:, 0], X.iloc[:, 1], y)
plt.xlabel("longitude")
plt.ylabel("latitude");
```

Real boundary between Canada and USA

In real life we know what's the boundary between USA and Canada.

In [ ]:
```
model = DecisionTreeClassifier(max_depth=1)
model.fit(X.to_numpy(), y)
plot_tree_decision_boundary_and_tree(
    model,
    X,
    y,
    height=6,
    width=16,
    eps=10,
    x_label="longitude",
    y_label="latitude",
)
```

In [ ]:
```
model = DecisionTreeClassifier(max_depth=2)
model.fit(X.to_numpy(), y)
plot_tree_decision_boundary_and_tree(
    model,
    X,
    y,
    height=6,
    width=16,
    eps=10,
    x_label="longitude",
    y_label="latitude",
)
```

Practice exercises

- If you want more practice, check out module 2 in this online course (https://ml-learn.mds.ubc.ca/en/module2). All the sections **without** video or notes symbol are exercises.

If all of you are working on the exercises, especially coding exercises, at the same time, you might have to wait for the real-time feedback for a long time or you might even get an error. There is no solution for this other than waiting for a while and trying it again.

Some background on the online course (https://ml-learn.mds.ubc.ca/en/) above: This course is designed by Hayley Boyce, Mike Gelbart, and Varada Kolhatkar. It'll be a great resource at the beginning of this class, as it give you a chance to practice what we learn and the framework will provide you real-time feedback.

Final comments and summary

What did we learn today?

- There is a lot of terminology and jargon used in ML. Some of the basic terminology includes:
  - Features, target, examples, training
  - Supervised vs. Unsupervised machine learning
  - Classification and regression
  - Accuracy and error
  - Parameters and hyperparameters
  - Decision boundary

- Baselines and steps to train a supervised machine learning model
  - Baselines serve as reference points in ML workflow.

- Decision trees
  - are models that make predictions by sequentially looking at features and checking whether they are above/below a threshold
  - learn a hierarchy of if/else questions, similar to questions you might ask in a 20-questions game.
  - learn axis-aligned decision boundaries (vertical and horizontal lines with 2 features)
  - One way to control the complexity of decision tree models is by using the depth hyperparameter (`max_depth` in `sklearn`).

In [ ]:
```
1
```